ADR : Architecture Decision Records

Documenter les décisions d'architecture de façon légère et durable

Stéphane FOSSE
fosse.fr

22 mars 2026

Résumé

Un Architecture Decision Record (ADR) est un fichier texte court, généralement en Markdown, qui documente une seule décision architecturale significative : le contexte qui l'a motivée, la décision elle-même, les alternatives écartées et les conséquences attendues. Introduits en novembre 2011 par Michael Nygard, ingénieur chez Cognitect, les ADR sont versionnés avec le code source dans le dépôt Git du projet. En 2017, le cabinet Thoughtworks les classe dans la catégorie *Adopt* de son Technology Radar, consacrant une pratique qui s'est depuis diffusée jusqu'aux équipes d'Amazon Web Services, du gouvernement numérique britannique et de l'Open Group.

Pourquoi les décisions architecturales disparaissent-elles ?

Tout projet vieillissant accumule une dette de compréhension autant qu'une dette technique. Un développeur arrivant six mois après le démarrage d'un système se retrouve face à des choix dont la motivation s'est évaporée avec les équipes qui les ont faits. Deux issues seulement s'offrent alors à lui : accepter aveuglément la décision, même si le contexte a changé, ou la renverser sans en mesurer les conséquences. Michael Nygard, dans son billet de novembre 2011 sur le blog de Cognitect [8], pose le problème sans détour : les grandes spécifications ne sont jamais lues ni maintenues, et les décisions importantes finissent dans les cerveaux des développeurs qui quittent l'équipe.

La racine du problème est structurelle. Les méthodes agiles valorisent le logiciel fonctionnel sur la documentation exhaustive, ce que Nygard approuve explicitement. Mais cette posture laisse un angle mort : la *motivation* derrière les choix architecturaux n'est ni du code ni de la documentation traditionnelle. C'est une troisième catégorie, ignorée par défaut. Michael Keeling, architecte chez IBM Watson, décrit dans son article paru dans *IEEE Software* en juin 2022 [4] cette tension de longue date entre la communauté de l'architecture logicielle et les équipes agiles : pendant des décennies, les architectes ont proposé des vues, des modèles, des notations — et les développeurs agiles les ont regardés avec une politesse distante. Les ADR ont changé cette dynamique parce qu'ils ne demandent ni outillage spécialisé, ni formation, ni processus lourd.

Qu'est-ce qu'un Architecture Decision Record (ADR) ?

Un ADR est un document d'une ou deux pages, rédigé en prose avec des phrases complètes. Nygard le compare aux patterns d'Alexander : chaque enregistrement décrit un ensemble de forces en tension et la décision prise en réponse à ces forces. Le format original définit cinq sections. Le **titre** est une courte phrase nominale qui identifie la décision (*ADR 9 : LDAP pour l'intégration multi-tenant*). Le **statut** indique si la décision est proposée, acceptée, dépréciée ou remplacée par un ADR ultérieur ; les numéros ne sont jamais réutilisés, et les ADR remplacés restent dans l'historique. Le **contexte** décrit factuellement les forces en présence : technologiques, politiques, sociales, contraintes de projet ; le ton est neutre, sans jugement. La **décision** est formulée à la voix active, au présent : « Nous utiliserons… ». Les **conséquences**, positives et négatives, clôturent le document ; elles deviennent souvent le contexte des ADR suivants, ce qui crée un enchaînement logique dans l'historique des décisions.

Ce format est délibérément minimal. Nygard indique que chaque ADR doit tenir en une ou deux pages. La brièveté n'est pas une contrainte arbitraire : elle garantit que le document sera réellement écrit, lu et maintenu. L'expérience de l'équipe WIRE chez IBM, relatée dans un rapport Agile Alliance de 2017 par Michael Keeling et

Joe Runde [5], confirme ce point : l'équipe de neuf développeurs travaillant sur le Watson Discovery Service a produit plus de 80 ADR en deux ans, traçant la naissance, l'évolution et la mise hors service de microservices. Le format s'est diffusé naturellement via le processus de pull request déjà utilisé pour les revues de code.

Quelle est la différence entre un ADR Nygard, un MADR et un Y-Statement ?

Trois formats dominent l'écosystème ADR, chacun répondant à un contexte différent. Le format **Nygard** reste le plus répandu pour sa simplicité. Il convient aux équipes qui démarrent avec les ADR et privilégient la légèreté sur l'exhaustivité. Son principal défaut est de ne pas structurer explicitement l'analyse des alternatives : un ADR Nygard décrit la décision retenue, mais documente rarement les options écartées avec leurs mérites et leurs défauts.

Le **Y-Statement**, formalisé par Uwe Zdun, Rafael Capilla, Huy Tran et Olaf Zimmermann dans *IEEE Software* en 2013 [14], compresse toute la décision en une phrase structurée : « Dans le contexte de <cas d'utilisation>, face à <contrainte>, nous avons décidé <option> pour atteindre <qualité>, en acceptant <inconvénient>. » Cette forme lapidaire convient aux contextes où les décisions doivent tenir sur une diapositive ou dans un commentaire de code. Elle force à identifier explicitement le compromis accepté, ce qui en fait un excellent outil pédagogique pour apprendre à raisonner en matière d'arbitrages architecturaux.

Le **MADR** (*Markdown Architectural Decision Records*), publié scientifiquement en 2018 par Oliver Kopp, Anita Armbruster et Olaf Zimmermann [7], est le format le plus complet. Il structure explicitement les options considérées avec leurs avantages et inconvénients respectifs, ajoute une section *Decision Drivers* pour identifier les forces prioritaires, et depuis sa version 3.0.0 en 2022 intègre une section *Confirmation* qui décrit comment vérifier que la décision a bien été mise en œuvre — par exemple via un test avec ArchUnit ou une revue de code ciblée. Le MADR version 4.0.0, publiée en septembre 2024, propose quatre variantes du template selon le niveau de détail souhaité : complet annoté, complet nu, minimal annoté, minimal nu.

Ces trois formats ne s'excluent pas. Plusieurs équipes utilisent le Y-Statement comme résumé de synthèse dans la section *Decision Outcome* d'un MADR, combinant la densité de l'un et la structuration de l'autre. L'organisation ADR sur GitHub, qui maintient adr.github.io depuis 2016, recense et compare ces formats ainsi qu'une vingtaine d'autres variantes issues de la communauté.

Où stocker les ADR dans un projet et comment les organiser ?

La convention la plus répandue, issue directement du billet de Nygard, est de placer les ADR dans le dépôt Git du projet sous `docs/adr/` ou `docs/decisions/`, numérotés séquentiellement (`0001-titre-en-tirets.md`). La proximité avec le code a une conséquence directe : les ADR suivent le même cycle de revue que les modifications de code, via des pull requests. Le UK Government Digital Service (GDS), qui utilise les ADR depuis plusieurs années pour ses projets numériques, recommande explicitement cette approche dans ses standards publiés [13], avec une mise à jour datée de mars 2026. Le GDS recommande également d'utiliser les *draft pull requests* de GitHub pour les ADR à l'état Proposé, ce qui rend le statut du document cohérent avec son état dans le workflow de développement.

Pour les projets multi-dépôts, AWS Prescriptive Guidance recommande de stocker les ADR transversaux dans le dépôt principal et de les référencer dans les README des projets dépendants [1]. Dans les grands projets qui accumulent des centaines d'ADR, le MADR propose d'organiser les fichiers en sous-répertoires correspondant à la structure architecturale du système (`decisions/backend/`, `decisions/frontend/`), au prix d'une numérotation locale plutôt que globale.

Un point que les équipes sous-estiment souvent : l'ADR n'est pas un document figé au moment de son écriture. Le GDS distingue trois phases de vie — Proposé, Accepté, Remplacé — et précise que les ADR acceptés mais non encore totalement déployés doivent faire l'objet d'un suivi dans l'outil de gestion de tickets. La décision n'est implémentée que quand elle l'est réellement dans tous les systèmes concernés.

Quand faut-il créer un ADR plutôt qu'une simple note ou un commentaire de code ?

La question de périmètre est probablement la plus délicate, et celle sur laquelle les équipes dérivent le plus souvent. Pierre Pureur et Kurt Bittner, dans un article publié par InfoQ en octobre 2023 [9], documentent ce

phénomène de glissement : faute de définition claire de ce qui est architectural, des équipes versent dans leurs ADR toute décision qu'elles jugent significative, transformant l'outil en un fourre-tout qui finit par noyer les vraies décisions architecturales.

Grady Booch fournit la formulation la plus utile : « Toute architecture est du design, mais tout design n'est pas de l'architecture. L'architecture représente l'ensemble des décisions de design significatives qui façonnent la forme et la fonction d'un système, où la signification se mesure au coût du changement. » Ce critère de coût ne suffit pas seul : changer l'interface graphique d'une application peut être long et coûteux sans être architectural. Ce qui distingue une décision architecturale, selon Pureur et Bittner, c'est que ses implications sont dispersées dans tout le code plutôt que localisées dans un module. Changer le paradigme de messagerie (synchrone vers asynchrone), la stratégie de consistance des données ou le modèle de sécurité au niveau de granularité des objets — voilà des décisions dont les conséquences se retrouvent dans des dizaines de fichiers. Choisir entre deux frameworks CSS ne l'est généralement pas.

Michael Nygard définit le périmètre par cinq types d'impact : la structure du système, les caractéristiques non-fonctionnelles, les dépendances entre composants, les interfaces exposées, les techniques de construction. Dans la pratique, une question simple permet de se décider : si cette décision est mal comprise dans six mois, quelqu'un va-t-il casser quelque chose en essayant de la contourner ?

Comment les ADR s'intègrent-ils dans une gouvernance d'architecture d'entreprise ?

Un ADR isolé a peu de valeur. C'est leur accumulation en un journal de décisions (*Architecture Decision Log*) qui crée de la valeur : la possibilité de retracer l'évolution du système, de comprendre pourquoi les choix actuels ont été faits, d'identifier les décisions qui devraient être réexaminées parce que leur contexte a changé.

Grygoriy Gonchar, dans un article InfoQ de mars 2023 [2], propose un cadre à trois niveaux pour la gouvernance architecturale. Le Technology Radar interne cartographie le paysage technologique de l'organisation. Les Standards technologiques définissent les règles transversales. Les ADR documentent enfin les décisions ponctuelles qui s'appliquent dans le cadre défini par les deux premiers niveaux. Ce découpage évite la surcharge des ADR tout en maintenant leur utilité : un ADR ne devrait pas avoir à argumenter pourquoi l'organisation utilise REST plutôt que XML-RPC si c'est défini dans les Standards.

Andrew Harmel-Law, dans son article publié sur martinfowler.com [3], pousse la logique encore plus loin. Pour des organisations avec des dizaines d'équipes autonomes, il propose l'*Advice Process* : n'importe qui peut prendre une décision architecturale, à condition d'avoir consulté au préalable toutes les personnes significativement affectées et les experts du domaine. L'ADR devient alors à la fois l'instrument de la consultation (un draft mis en revue) et la trace de la décision prise. Cette approche distribue l'autorité architecturale sans la dissoudre, parce que le processus de consultation et la documentation des conséquences maintiennent la rigueur.

L'Open Group, dans sa documentation sur l'architecture intentionnelle publiée dans le cadre du standard ouvert *Open Agile Architecture* [11], positionne les ADR dans les trois axes de l'architecture TOGAF : architecture d'entreprise (horizon long terme, niveau entité), architecture de solution (périmètre projet) et architecture technique. Les ADR sont pertinents aux trois niveaux, avec une granularité et un cycle de vie différents selon l'altitude.

Quels outils existent pour gérer les ADR ?

L'outillage est délibérément minimal, ce qui fait partie de l'attrait de la pratique. Nat Pryce a publié en 2016 `adr-tools`, un ensemble de scripts shell pour initialiser un répertoire ADR, créer un nouvel enregistrement, marquer un ADR comme remplacé par un autre et générer un sommaire. La commande
`adr new "Utiliser PostgreSQL pour les données transactionnelles"`
crée le fichier
`0005-utiliser-postgresql-pour-les-donnees-transactionnelles.md`
avec le template pré-rempli et le numéro suivant dans la séquence.

Pour les équipes sur GitHub ou GitLab, le processus de pull request suffit comme workflow de validation. Un ADR Proposé correspond à une PR ouverte ; un ADR Accepté correspond à une PR mergée sur la branche principale. Le GDS recommande cette convention implicite pour éviter la redondance entre le statut inscrit dans le fichier et l'état de la PR. Des extensions Visual Studio Code existent pour le MADR, bien que la documentation du projet signale que la version disponible sur le marketplace peut ne pas supporter les dernières fonctionnalités de MADR 4.x.

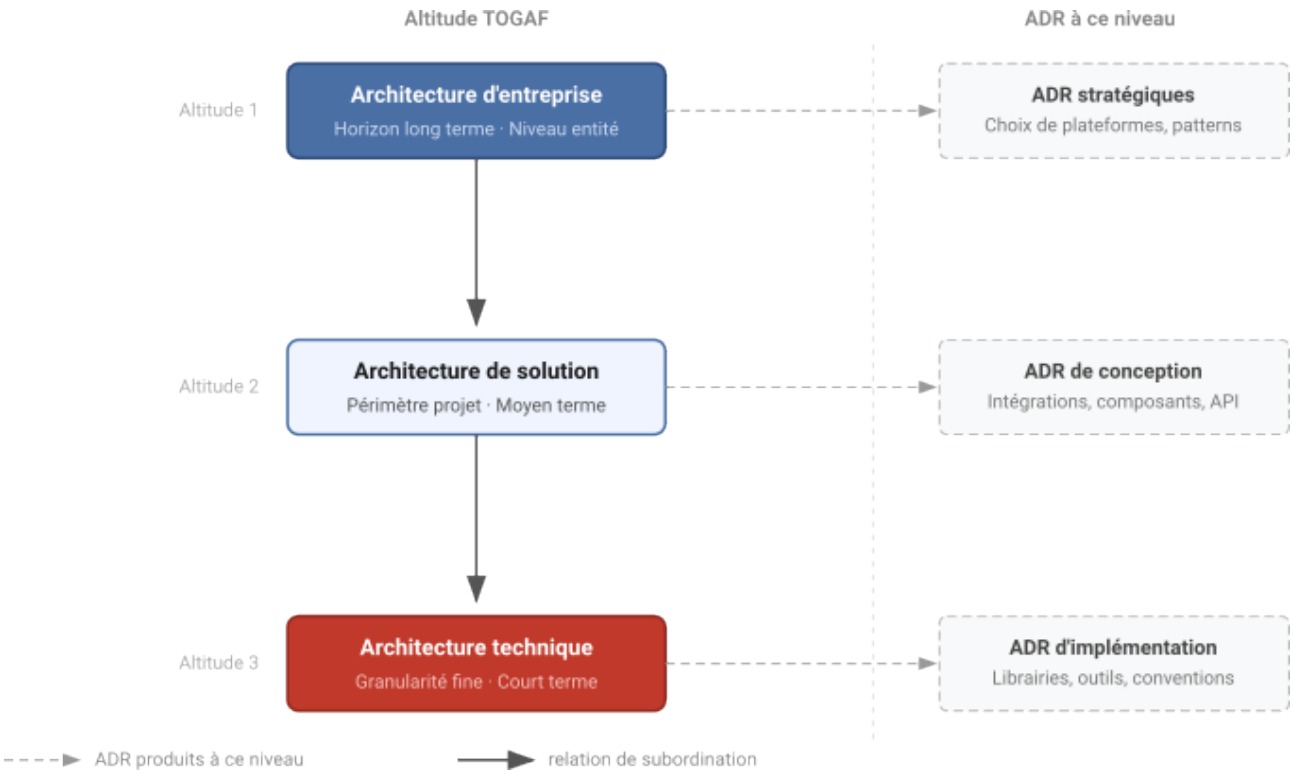

FIGURE 1 – ADR aux trois altitudes TOGAF (Open Group, Open Agile Architecture)

Azure Well-Architected Framework référence les ADR depuis octobre 2024. AWS fournit depuis plusieurs années un guide prescriptif complet qui couvre le cycle de vie des ADR, les rôles, le processus de revue et les bonnes pratiques — notamment la traçabilité entre ADR et code, et la gestion du code legacy non conforme aux décisions documentées.

Conclusion

Les ADR résolvent un problème que les équipes ont longtemps ignoré parce qu'il est invisible : la perte de la raison d'être des décisions techniques. Ce n'est pas de la documentation pour la documentation. C'est la mémoire du raisonnement collectif, versionnée avec le code qui en est la conséquence. Leur adoption depuis 2017 dans des contextes aussi différents que le gouvernement numérique britannique, IBM Watson, les architectures cloud AWS et des centaines d'équipes open source témoigne d'une convergence rare dans notre industrie : une pratique simple, sans outillage lourd, qui tient ses promesses à l'usage.

La vraie difficulté n'est pas technique. Elle est culturelle : écrire un ADR suppose d'accepter de rendre visible son raisonnement, y compris les alternatives qu'on a écartées et les compromis qu'on a acceptés. Pour un architecte, c'est aussi exposer ses doutes. C'est précisément pour ça que ça marche : un ADR honnête, qui liste les inconvénients de la décision retenue, est bien plus utile qu'un document qui justifie après coup un choix déjà fait. Et c'est ce que les équipes qui les pratiquent finissent par découvrir.

Références

[1] AMAZON WEB SERVICES. Using Architectural Decision Records to Streamline Technical Decision-Making for a Software Development Project. Anglais. AWS Prescriptive Guidance. Guide prescriptif complet couvrant le cycle de vie des ADR. 2024.

[2] Grygoriy GONCHAR. A Simple Framework for Architectural Decisions. Anglais. InfoQ. Cadre à trois niveaux : Technology Radar, Standards technologiques et ADR. Mars 2023.

[3] Andrew HARMEL-LAW. Scaling the Practice of Architecture, Conversationally. Anglais. martinfowler.com. L'Advice Process comme mécanisme de gouvernance architecturale distribuée. 2021.

[4] Michael KEELING. Love Unrequited: The Story of Architecture, Agile, and How Architecture Decision Records Brought Them Together. Anglais. In : *IEEE Software* 39.4 (2022). The Pragmatic Designer column. Analyse la tension entre architecture et agilité, et le rôle des ADR., p. 90-94.

[5] Michael Keeling et Joe Runde. Distribute Design Authority with Architecture Decision Records. Anglais. In : *Agile Alliance Experience Reports*. Retour d'expérience de l'équipe WIRE chez IBM Watson Discovery Service (80+ ADR sur deux ans). 2017.

[6] Oliver Kopp, Anita Armbruster et Olaf Zimmermann. About MADR — Markdown Architectural Decision Records. Anglais. Documentation officielle du format MADR, version 4.0.0 (septembre 2024). 2024.

[7] Oliver Kopp, Anita Armbruster et Olaf Zimmermann. Markdown Architectural Decision Records: Format and Tool Support. Anglais. In : *Proceedings of the 10th ZEUS Workshop on Services and their Composition (ZEUS 2018)*. Publication scientifique décrivant la genèse du format MADR. 2018.

[8] Michael Nygard. Documenting Architecture Decisions. Anglais. Cognitect blog. Le billet fondateur qui formalise le concept d'ADR. Nov. 2011.

[9] Pierre Pureur et Kurt Bittner. Has Your Architectural Decision Record Lost Its Purpose? Anglais. InfoQ. Analyse des dérives fréquentes dans l'utilisation des ADR. Oct. 2023.

[10] Heiko W. Rupp. Why you should be using architecture decision records to document your project. Anglais. Red Hat Blog. Introduction pratique aux ADR dans un contexte open source. Nov. 2025.

[11] The Open Group. Intentional Architecture — Open Agile Architecture Standard. Anglais. Positionnement des ADR dans le cadre TOGAF et les trois axes de l'architecture intentionnelle. 2021.

[12] Thoughtworks. Lightweight Architecture Decision Records. Anglais. Technology Radar, catégorie Adopt, novembre 2017. 2017.

[13] UK Government Digital Service. Documenting Architecture Decisions — The GDS Way. Anglais. Standard institutionnel du gouvernement numérique britannique, mis à jour mars 2026. 2026.

[14] Uwe Zdun et al. Sustainable Architectural Design Decisions. Anglais. In : *IEEE Software* 30.6 (2013). Republié sur InfoQ. Introduit le Y-Statement et les bases théoriques des décisions durables., p. 46-53.